J Real-Time Image Proc (2016) 11:111–125
DOI 10.1007/s11554-012-0310-5

ORIGINAL RESEARCH PAPER

Parallel implementation of background subtraction algorithms for real-time video processing on a supercomputer platform

Grzegorz Szwoch · Damian Ellwart · Andrzej Czyżewski

Received: 27 April 2012 / Accepted: 6 December 2012 / Published online: 30 December 2012

Abstract Results of evaluation of the background subtraction algorithms implemented on a supercomputer platform in a parallel manner are presented in the article. The aim of the work is to chose an algorithm, a number of threads and a task scheduling method, that together provide satisfactory accuracy and efficiency of a real-time processing of high-resolution camera images, maintaining the cost of resources usage at a reasonable level. Two selected algorithms: the Gaussian mixture models and the Codebook, are presented and their computational complexity is discussed. Various approaches to the parallel implementation, including assigning the image pixels to threads, the task scheduling methods and the thread management systems, are presented. The experiments were performed on a supercomputer cluster, using a single machine with 12 physical cores. The accuracy and performance of the implemented algorithms were evaluated for varying image resolutions and numbers of concurrent processing threads. On a basis of the evaluation results, an optimal configuration for the parallel implementation of the system for real-time video content analysis on a supercomputer platform was proposed.

Keywords Background subtraction · Parallel computing · Object detection · Gaussian mixture models · Codebook

G. Szwoch (✉) · D. Ellwart · A. Czyżewski
Multimedia Systems Department, Gdansk University of Technology, Narutowicza 11/12, 80-233 Gdańsk, Poland
e-mail: greg@sound.eti.pg.gda.pl

D. Ellwart
e-mail: ellwart@sound.eti.pg.gda.pl

A. Czyżewski
e-mail: andcz@sound.eti.pg.gda.pl

1 Introduction

Background subtraction is a procedure commonly used for separation of the image pixels belonging to moving objects from those representing a static background. This is one of the most frequently used algorithms in video content analysis frameworks for a detection and tracking of moving objects and an automatic event detection [5]. At the same time, it is one of the most computationally intensive image analysis procedures. Video monitoring systems for the automatic event detection require a real-time background subtraction, which is difficult to achieve with high-resolution cameras often employed in the modern video surveillance systems. A solution to this problem is either to reduce the image resolution or/and frame rate, which decreases the accuracy of the background subtraction, or to utilize parallel processing methods, dividing the task of image analysis into several concurrently running processing threads.

Recently, the authors faced a problem of implementing a background subtraction algorithm within a parallel data processing framework running on a supercomputer—a multi-node system of multi-core machines. The framework named KASKADA was designed for efficient processing of high amounts of data from multiple sources [18]. In the context of video monitoring, KASKADA will be used for concurrent analysis of the images obtained from a large number of cameras, for identifying specified objects or safety threads. The chain of the processing algorithms is run on multiple nodes of the supercomputer that communicate with each other using a message passing interface. The background subtraction algorithm is one of the initial processing stages: it receives the decoded camera images, performs the background subtraction and sends the results to further video analysis algorithms (object detection and

tracking, automatic event detection, analysis of the crowd behavior, etc.), running on other nodes of the supercomputer. Multiple instances of the background subtraction algorithm have to run concurrently on separate nodes, analyzing a large number of video streams in the real time.

The implementation of a background subtraction algorithm within the KASKADA framework required answering several important questions: (1) which background subtraction algorithm is the most suitable for parallel implementation in this framework, (2) which thread management strategy is optimal, (3) how many computing units are required for real-time processing and (4) which algorithm is optimal in terms of balance between its performance and the accuracy. The aim of this article is to address these questions using a set of tests performed using the real computing system. We assumed that in a production system, a single instance of the background subtraction algorithm will be run on a single node of the supercomputer and such a solution was implemented and tested.

A variety of methods for background estimation can be found in the literature [13, 15, 16, 26]. Among all of them, the approach based on Gaussian mixture models (GMM) seems to be the most popular one. The algorithm representing the background model of a single pixel as a set of weighted Gaussian distributions was first proposed by Friedman and Russell [9], and later extended by Stauffer and Grimson [29] who proposed an efficient method of model updating. Numerous further improvements of the original GMM method were proposed. KaewTraKulPong and Bowden [12] used an expectation maximization approach for improving the learning rate of the background model. Setiawan et al. [25] applied the GMM method to an improved hue-luminance-saturation color space in order to achieve better sensitivity to color changes. An important work by Zivkovic and Van der Heijden [33] resulted in an improved adaptation of the GMM model to changes in the analyzed scene by automatic selection of the number of Gaussian components. On this, basis further work was carried out by Sicre and Nicolas [27] to further improve the model adaptation capabilities. Additionally, a modification of the GMM model adaptation process leading to reduction of the object detection errors in case of illumination variations is proposed by the authors of this work. An important problem in the background subtraction is the shadow removal, as moving shadows are assigned to the foreground in the original GMM method, so an additional procedure for removing the shadows is required. An algorithm proposed by Horprasert et al. [10] based on analysis of the color and brightness variations is often used for this task.

Another, less popular approach to the background subtraction is the Codebook algorithm proposed by Kim et al. [16]. This method represents a pixel in the background model using a set of codewords describing the color, brightness and statistical properties of the pixels. According to its author, this method outperforms the GMM algorithm in terms of handling moving backgrounds and illumination variations, as well as in computational complexity. Kim et al. [17] improved this method for more robust background maintenance. Li et al. [19] introduced Gaussian distributions to the Codebook algorithm for quantization of the temporal series in order to reduce the rate of false-negative results. Sigari and Fathy [28] presented a two-layer implementation of the Codebook model for handling background changes. Ilyas et al. [11] improved maintenance of the codebook background model.

Several other background subtraction algorithms may be found in the literature. A survey of such methods may be found, e.g.. in works of Parks and Fels [20] or Benezeth et al. [1]. This article focuses on two chosen algorithms: the GMM, which is the most widely used, and the Codebook, because the published test results are promising.

A published work on the parallel implementation of background subtraction algorithms is mainly limited to the GPU computing area, as presented, e.g., by Fauske et al. [8] or Pham et al. [23]. The implementation of the algorithm on a parallel system with a low number (up to 12) of processing units requires a proper thread management and a task scheduling approach. The Boost Threads library is a commonly used solution for the platform-independent thread management controlled by the programmer [14]. An alternative is the automatic task scheduling, managed by an external library. The most popular solutions of this type are libraries based on the Open Multi-Processing (OpenMP) application programming interface for the shared memory multiprocessing computing [3] and the Thread Building Blocks (TBB) library [24].

The article is organized as follows. First, both the GMM and the Codebook algorithms are described. The computational complexity of both methods, as well as strategies for their parallel implementation are discussed. Next, the results of experiments in which the performance and accuracy of both algorithms were tested in different conditions are presented. The conclusions and the discussion on the practical implications of the obtained results conclude the article.

2 Background subtraction algorithms

The purpose of background subtraction algorithms is to divide all the image pixels into two groups: the foreground pixels, representing moving objects, and the background pixels, belonging to the static background. This separation is necessary in order to select the pixels representing the actual moving objects for the purpose of object detection,

tracking, etc. [5]. The background modeling is usually performed by constructing a background model, which may be based, e.g., on the statistical analysis of the pixel values. The actual subtraction is achieved by comparing the current pixel values with the model and making a binary foreground/background decision. The shadows of the moving objects need to be eliminated from the foreground pixels, either by the subtraction algorithm itself or by using a separate post-processing method. The result of the background subtraction usually needs to be post-processed, e.g., with the morphological operations, in order to clean the resulting binary mask (remove the noise and fill small holes). The details of the two background subtraction algorithms chosen for the evaluation are presented below.

2.1 Gaussian mixture model

The Gaussian mixture model proposed first by Friedman and Russell [9] is a probabilistic method used for the background modeling. This approach is based on the assumption that upon the observations made to an image pixel, the associated background representation can be chosen as the most frequent appearing value. As this esteem can fluctuate, even under the strictly controlled conditions, e.g., due to the image noise, the background model of each pixel is described by a Gaussian given by:

$$\eta(X,\mu,\Sigma) = \frac{1}{(2\pi)^{0.5D}|\Sigma|^{0.5}} \mathrm{e}^{-0.5\left[(X-\mu)^T\Sigma^{-1}(X-\mu)\right]}, \quad (1)$$

where μ denotes the mean of the distribution, Σ represents the covariance matrix and X stands for the pixel value. For simplification, it is assumed that the color channels are independent, then $\Sigma = \sigma^2 \cdot I$, where σ is the standard deviation of the distribution. Since the real life background varies during the day and night, the Gaussian adaptation is introduced in order to handle these changes. The formula for the parameters update is given by:

$$\mu_t = (1-\rho)\cdot\mu_{t-1} + \rho X_t \quad (2)$$

$$\sigma_t^2 = (1-\rho)\cdot\sigma_{t-1}^2 + \rho(X_t-\mu_t)^T(X_t-\mu_t), \quad (3)$$

where μ_t and σ_t are the mean and the standard deviation of the distribution, ρ is the update factor and X_t denotes the processed pixel value. This way, the background model is adjusted to reflect changes in the analyzed video content. In order to increase the model adaptation capabilities, each image point is characterized by a mixture of (typically 3–5) Gaussians. A number of Gaussians can either be constant (defined when the algorithm is run) or it can vary depending on the scene characteristics [25]. Regardless of the implementation, this approach allows for handling situations such as when an object is left or taken from the scene. Although multiple distributions are utilized, only one of them represents the current model. Hence, weight factors are assigned to each of the Gaussians to indicate their strengths. A weight adaptation procedure is defined as:

$$\omega_{k,t} = (1-\alpha)\cdot\omega_{k,t-1} + \alpha M_{k,t}, \quad (4)$$

where $\omega_{k,t}$ is the weight of the kth distribution, α is the learning factor and $M_{k,t}$ is a binary value equal to 1 if the current pixel matched a distribution and 0 otherwise. In order to verify whether the currently analyzed pixel should be assigned to the background or foreground, its value is tested against the distributions ordered by a descending ω_k/σ_k factor. Typically, if the value fits the leading distribution in the range of 2.5 standard deviations (defined as the Δ parameter) from the Gaussian mean, the point is marked as belonging to the background. Otherwise, the pixel value is used to form a new distribution, which replaces the one with the lowest ω_k/σ_k factor.

The detection accuracy obtained utilizing this method is satisfactory in many cases. However, in the real life scenarios, changing lighting conditions can often cause the detection errors. It can be observed especially for outdoor scenes with high illumination variations. Therefore, further algorithm improvements were developed to solve this problem.

Detection errors for scenes with high illumination variations are related to the distribution adaptation process, which cannot update the model sufficiently fast to compensate the changes. This could be solved by applying a variable learning factor [30]. On the other hand, such a modification can also cause the detection errors. Higher ρ values affect the distribution mean and deviation adaptation rate. Hence, in case of long-term lighting variations, the leading Gaussian can be discarded from its position (considering the ω/σ factor) due to the increasing deviation. This problem can be partially solved by utilizing independent adaptation factors for Gaussian mean and deviation [4]. Another modification of the GMM introduces spatial dependencies for the pixel assignment process, making it more robust [32]. On the other hand, KaewTraKulPong and Bowden [12] proposed different adaptation formulas for various processing stages to improve the initial model learning process.

The GMM modification proposed in this article applies to scenes with lighting variations. It is based on the observation that the illumination changes are smooth considering the inter-frame differences. For such conditions, the background modeling algorithm should have high adaptive capabilities. This is achieved by applying an additional post-processing stage where particular model regions are updated. To determine these regions, the pixel variability is estimated as a mean differential of the consecutive video frames in which no objects are detected:

$$\mathrm{diff}_n = (1-\gamma)\mathrm{diff}_{n-1} + \frac{1}{C}\gamma \sum_c \left| I_c(x,y)_n - I_c(x,y)_{n-1} \right|, \tag{5}$$

where $I(x,y)$ denotes the image point at (x,y) coordinates, γ is the pixel variance learning factor in the range (0, 1) and C is the number of image color channels. This way, a matrix of values is created, size of which is equal to the analyzed image resolution. Next, each value of this matrix is thresholded (D_{th}) in order to determine regions characterized by a low variability. For the pixels related to the background, which fulfill this condition, an additional update process is performed. This is acquired by shifting (with a set weight) the leading distribution mean toward the actual value from the input frame. This adaptation does not change the variation of the Gaussian, hence the ω/σ factor for the modified distribution is constant. Due to improved adjustment capabilities of the leading distribution, it is referred to as a short-term model further on. In order to reduce local errors which can be caused by objects with the color similar to the background model, an additional Gaussian is utilized. It is considered as a long-term background representation. Adaptation of this distribution is carried out utilizing the relations described by Eqs. (2) and (3). Another advantage of this approach is that it allows for reduction of the detection sensitivity in case of scene illumination variations. It is achieved by utilizing the short- and long-term models at the same time. Since the long-term representation is independent from the regular model adaptation procedure, it can cover the same value ranges as the k Gaussians. Hence, in case of low image variability, the short- and long-term distributions can be similar. However, if lighting changes are present in the scene, the short- and long-term Gaussians spread, resulting in the sensitivity reduction.

2.2 The Codebook algorithm

The Codebook method was proposed by Kim et al. [16] for the real-time foreground–background segmentation. The author claims that this method is efficient in speed compared to the other background subtraction methods and it is more robust in scenes with illumination variations. However, so far this algorithm has not been used in video analysis systems as widely as the GMM method. The background model in the Codebook algorithm is based on representing each image pixel by a number of codewords [16]. A single codeword represents a range of color and brightness variations of the pixel and is described using a vector of nine parameters:

$$c_i = \left\langle \bar{R}_i, \bar{G}_i, \bar{B}_i, \breve{I}_i, \hat{I}_i, f_i, \lambda_i, p_i, q_i \right\rangle \tag{6}$$

where $(\bar{R}_i, \bar{G}_i, \bar{B}_i)$ are the pixel values in the RGB color space, $\left(\breve{I}_i, \hat{I}_i\right)$ define the lower and upper range of the brightness variations, f_i counts a number of times the codeword was matched, λ_i is the maximum negative run-length (MNRL), defined as a longest interval in which the codeword has not recurred, p_i and q_i store the first and the last access time of the codeword, respectively.

In the original Codebook algorithm, the background model has to be built using a number of initial image frames before the actual background subtraction may be performed. During this initial training phase, the pixel value in each image frame is compared with all the codewords representing this pixel in the background model. If a matching codeword is found, it is updated, otherwise a new codeword is created and added to the background model. A codeword $\mathbf{c}_i$ matches a pixel (R, G, B) if two conditions are fulfilled. The first condition describes the color difference between the pixel and the codeword and is given by

$$\sqrt{(R^2+G^2+B^2) - \frac{(R\bar{R}+G\bar{G}+B\bar{B})^2}{\bar{R}^2+\bar{G}^2+\bar{B}^2}} \le \varepsilon, \tag{7}$$

where ε is a constant threshold that defines the maximum allowed color variation. The second condition is related to the brightness variations and it is introduced in order to take the illumination changes and shadows into account. The condition is given by:

$$\alpha\hat{I} \le \sqrt{R^2+G^2+B^2} \le \min\left\{ \beta\hat{I}, \frac{\breve{I}}{\alpha} \right\}, \tag{8}$$

where α and β are constants (the shadow and the highlight threshold, respectively) that limit the range of the pixel brightness variations.

If a matching codeword is found according to Eqs. (7) and (8), its (R, G, B) values are updated with the current pixel values, using a running average. The brightness range of the codeword is extended if the pixel brightness was outside this range. The statistic parameters of the codeword (the update times and MNRL) are updated accordingly. If none of the codewords representing the pixel were matched, a new codeword is created and initialized with the current pixel values and the current frame number, then the codeword is added to the background model [16]. Therefore, during the training phase, new codewords are added each time the color or the brightness of the pixel change considerably.

After the training period is finished, the background model is pruned by eliminating the codewords with MNRL exceeding the threshold, which is typically equal to half of the training time. Then, the detection phase is performed simply by finding the matching codeword using the two

conditions described before. If a match is found, the pixel is assigned to the background and the matching codeword is updated. If no match is found, the pixel is assigned to the moving object, but no codeword is added to the model. This approach has a drawback: the background model does not adapt to the scene and illumination changes. Therefore, a more complex approach to the background subtraction, utilizing the method proposed by Kim is used for the background maintenance [17]. Two layers are added to the background model, already containing the permanent background layer obtained during the training: a long-term background (the codewords added to the model after the training is finished) and a cache (the codewords created during the detection phase, but not added to the background). Therefore, the modified detection procedure is organized as follows.

- The search for a matching codeword is performed in all model layers.
- If a match was found in the permanent or the long-term model layer, the pixel is assigned to the background.
- If a matching codeword was not found or the only match was found in the cache, the pixel is assigned to the moving object (foreground).
- The matching codeword, if found, is updated.
- If no matching codeword was found, a new codeword is created and put in the cache.
- The codewords in the cache that were not updated during the time T_{h} are removed.
- The codewords that remain in the cache longer than the time T_{add} are moved to the long-term model layer, from this time on, they represent the background.
- The codewords in the long-term layer that were not updated for a defined time T_{del} (usually much longer than T_{h}) are removed from the model.

With this procedure, adaptation of the background to the scene and illumination changes is achieved, at a cost of increased computational complexity. For the proper adaptation of the model to varying conditions, the threshold parameters need to be adjusted according to the character of variations.

In the original Codebook method, the background subtraction during the training phase is not possible. This is a drawback in comparison to the GMM method. Therefore, we modified the training phase by selecting the matched codewords with MNRL lower than the half of the elapsed training time as those representing the background, and the remaining ones as representing the moving objects. With this modification, a coarse background subtraction may be performed during the training phase. Additionally, in order to reduce the memory usage, the codewords are not added to the model if the elapsed training time is larger than the maximum allowed MNRL, since these codewords would be removed after the training anyway.

2.3 Morphological processing

The background subtraction result obtained using any of the described methods is a binary mask with the background and foreground pixels encoded using different values (usually 0 for the background and 255 or 1 for the foreground). The raw binary mask obtained from the background subtraction usually requires a post-processing before it may be used by the following video analysis algorithms. Due to imperfections of the background subtraction algorithms, the mask is distorted by the noise—single pixels or small groups of pixels being the false-positive results of the algorithm—and holes—a similar effect for the false-negative results. A common method of removing this noise and holes is using a morphological processing [6]. Two typical morphological operations used for this task are the dilation which increases the number of foreground pixels and closes small holes, and the erosion—a dual operation that reduces the number of foreground pixels and removes the noise. Since these operations distort the background subtraction result, they need to be performed together in a sequence. Typically, first the morphological opening (the erosion followed by the dilation) is performed for removing the noise, then the morphological closing (the dilation followed by the erosion) for closing small holes. All the morphological operations are usually performed using the same structuring element. In the presented work, a 3×3 pixels structuring element was used. The cleaned mask is passed as an input to the next processing stage, which usually performs the mask segmentation and extracts the connected components representing the moving objects. At this stage, groups of the false-positive pixels are removed by imposing a limit on the minimum connected component area.

3 Analysis of the computational complexity

There are two factors determining the time needed to process a single image frame with the background subtraction algorithm. The first one is the image resolution—the number of pixels to process. Assuming that no image pixels are removed from processing (masked out), all the pixels have to be processed independently, so the increase in image resolution results in a prolonged processing time (the relation should be linear in case of a sequential processing). The second, more important factor is the image content variability, which affects the processing time of a single pixel. If the scene in the sequence of images is 'empty' and stable, the processing time for each pixel

should be identical (provided that a stable background model is already constructed). In practical situations, the motion occurs only in parts of the image and the changeability of the image content is usually not uniform—there are stable regions and the parts with different rate of content changes. Some of these changes may be recurring, other may be the short-term ones. This has a serious impact on the processing time of a single pixel.

3.1 The GMM algorithm

Complexity of the GMM method depends on many factors. It is assumed that color images are processed and the model for each of the image pixels is already created. The processing time T is defined as:

$$T = T_M + kT_U, \tag{9}$$

where T_{M} represents the time needed to find a matching Gaussian, k is the number of distributions, and T_{U} is the time required for the model update process. During the matching stage (corresponding to the T_{M} time), each image pixel is compared with k Gaussians. To assess whether the current pixel fits a particular distribution, its value is compared with the mean and the standard deviation of this distribution. This process is repeated for k existing distributions, until a match is found. In order to improve the performance, the Gaussians are typically sorted in the order of decreasing ω/σ factor. Hence, the most probable match can be acquired in the first iteration. In case the analyzed pixel does not match any of the Gaussians, a new distribution is created replacing one of the existing, which is characterized by the lowest ω/σ factor.

The second GMM processing stage (corresponding to T_{U} time) is related to the model update procedure. First, for the matching Gaussian, its parameters are adjusted according to Eqs. (2) and (3). Additionally, the resulting Gaussian variance is tested against a set minimum value. Next, regardless of whether a match was previously found or a new distribution was created, the distributions weights are updated according to Eq. (4).

The described procedure is carried out for each image pixel. However, the time required to analyze different image regions can vary significantly due to various scene characteristics. The more frequent movement can be observed in the image, the more difficult it is to model the background. Hence, more operations are required to be carried out, extending the processing time. This generally leads to increased T_{M} time, since T_{U} depends mostly on the number of Gaussian distributions. The modifications introduced to the original GMM method require extra calculations to be performed. However, in experiments with the adjusted adaptation method, fewer distributions are utilized (three instead of five). This compensates the time needed to carry out the additional update stage. Hence, both these methods are considered as one during the efficiency evaluation. Additional operations are needed to detect the shadowed regions falsely recognized as objects in the object detection stage.

3.2 The Codebook algorithm

In the Codebook algorithm, the total processing time of a single image pixel is a sum of the times needed to find a matching codeword (if any) and updating the matching codeword (if it was found) or creating and adding a new codeword to the model (in case a match was not found). Because the creation of a new codeword is simple, this factor was excluded from the analysis. The times needed for the memory allocation and restructuring of the codebook (removing the stale codewords, moving the codewords between the layers, etc.) were also omitted.

Searching for a matching codeword requires testing the brightness and color conditions for each codeword in the model, until a codeword with the matching color and brightness is found in any layer of the model (either the permanent or the long-term one, but not in the cache). Since the color condition described by Eq. (7) requires performing a considerably larger number of operations than the brightness condition expressed by Eq. (8), searching for a matched codeword is started with the brightness condition and, if it is fulfilled, the color condition is tested next. Because both the conditions have to be fulfilled in order to match the codeword, the non-matching codewords are discarded from the further analysis after the simpler condition was not met, which reduces the overall computation time.

There are two important factors that affect the analysis time of a single image pixel. First, the number of codewords representing the pixel is not constant in time, it depends on the image changeability and the rates of the codebook updating, from a single codeword in the stable regions to five or even more codewords in the highly variable regions (especially when the codebook cleaning is performed rarely). This is an important difference compared to the original GMM method in which the number of distributions per pixel is constant. Second, the analysis time of a single codeword depends on the number of codewords for which testing of both conditions was necessary. Therefore, the processing time in different regions of the image may differ significantly. If the time needed for testing the brightness and the color condition are denoted as T_{B} and T_{C}, respectively, and the update time is T_{U}, the total processing time for a single image pixel can be expressed as:

$$T = N_{\mathrm{B}}T_{\mathrm{B}} + N_{\mathrm{C}}(T_{\mathrm{B}} + T_{\mathrm{C}}) + N_{\mathrm{U}}(T_{\mathrm{B}} + T_{\mathrm{C}} + T_{\mathrm{U}}), \tag{10}$$

where N_{B} is the number of codewords rejected after testing the brightness condition, N_{C} is the number of codewords

rejected after testing both conditions, N_U is one if a matching codeword was found (so that it needs updating) and zero otherwise. In order to reduce the processing time, a matched codeword is moved to the front of the codebook, so that it is tested as the first one in the next image frame (because this codeword is a most likely match for the next image).

As a practical example of the relations discussed above, a camera view consisting of two regions will be presented. One of these regions is stable, e.g., a wall of a building in a constant lighting conditions, while the other is constantly changing, e.g., a busy sidewalk with people in different color clothing, moving continuously. In the first region, a matching codeword will be usually found and updated in the first step, so $T_1 = (T_B + T_C + T_U)$. In the second region, the processing time will be usually larger than T_1, but it cannot be predicted. The second codeword may be matched (in case of a recurrent movement) or none of the codewords may match. In the worst case, all the codewords may be rejected after testing both conditions, so that $T_{max} = N \cdot (T_B + T_C)$, where N is the number of codewords representing the pixel. This may happen if a number of objects of a similar brightness but different color move through the observed region, so that a large number of codewords is created.

The observation that the processing time depends on variability of the image content has serious implications on load balancing in the parallelized algorithm. This problem will be discussed further in the article.

4 Parallelization strategies

The background subtraction algorithms described earlier—the GMM and the Codebook—perform independent analysis of each pixel, i.e., the result of analysis of a given pixel does not depend on the analysis result of any other pixel. Therefore, both algorithms are suitable for implementation in Single Instruction Multiple Data systems, as well as for a multi-threaded, parallel processing. In a massively parallel approach, each pixel could be analyzed in a separate thread. While this approach may be suitable for GPU computing, it is impractical in systems with a limited number of processing units. Therefore, the image is typically divided into several parts and each part is assigned to a processing thread. The main problem is a proper choice of the thread management and task scheduling methods. Four possible choices are presented in Table 1.

Two approaches to the thread management are possible: a manual and an automatic one. In the manual method, the programmer creates and manages the threads, e.g., using the Boost Threads library [14], and controls the task scheduling (assigning the image sections to the processing threads). This approach has an advantage of a complete control over the parallel processing. The automatic method utilizes a framework for parallel processing, e.g., using the OpenMP API [3], which performs the thread management and task scheduling using the black-box approach. This method is easier to implement, but the programmer loses the complete control over the process.

The task scheduling may be done using different approaches. Sections of the image may be assigned to the threads a priori, before the processing is started, e.g., the image may be cut into four horizontal stripes and each part is processed by a separate thread. An alternative approach is to assign smaller chunks of data to threads, e.g., a single row of pixels may be initially given to each thread. When a thread finishes its work, it receives another row of pixel to process. This procedure is repeated until the whole image is processed. This approach should, in theory, provide a better work balancing between threads, at a cost of overhead related to more complex thread management. In the OpenMP API, these two approaches are named the static and the dynamic task scheduling, respectively [3]. The dynamic approach is easy to implement in the automatic thread management, in OpenMP it only requires adding a single pragma command to the code. Using the dynamic method with the manual thread management is possible, but requires additional work needed for creating an own thread management system. Figure 1 presents simplified block diagrams of both task scheduling methods.

As it was discussed previously, in both background subtraction algorithms, especially in the Codebook method, the processing time of a single pixel varies according to the rate of image changeability. This is important when a parallelization strategy is selected. The most intuitive approach is to use manually managed threads with the static task scheduling, e.g., to create four threads and assign a horizontal image strip for each thread. However, let us discuss what happens if one of these strips contains changing content (e.g., a busy street) and the others are static (the sky, buildings, a lawn, etc.). If the static approach is used, the thread which receives the 'street' part will have to work longer (more codewords or Gaussians to check) and the other threads will finish their work earlier and they will be idle, resulting in a poor work balance. Therefore, the dynamic approach seems to be more suitable for the background subtraction. The preliminary experiments showed that even in case of more uniform movement in the image, the dynamic approach was more efficient than the static one [7]. This is further verified in the experiments described in this article.

OpenMP was employed for the task scheduling in the background subtraction, because it allows for selection of either the dynamic or the static scheduling method. Since the processing algorithm analyzes the image by rows (the

Table 1 Possible implementation of thread management and task scheduling systems for parallel background subtraction in video

	Thread management	
	Manual	Automatic
Task scheduling		
Static	Threads managed by the programmer, e.g., with *boost::threads*	Pre-allocation of image parts, OpenMP: *#pragma parallel for schedule (static)*
Dynamic	As above + task scheduling system written by the programmer	Image rows assigned to threads on demand by OpenMP: *#pragma parallel for schedule (dynamic)*

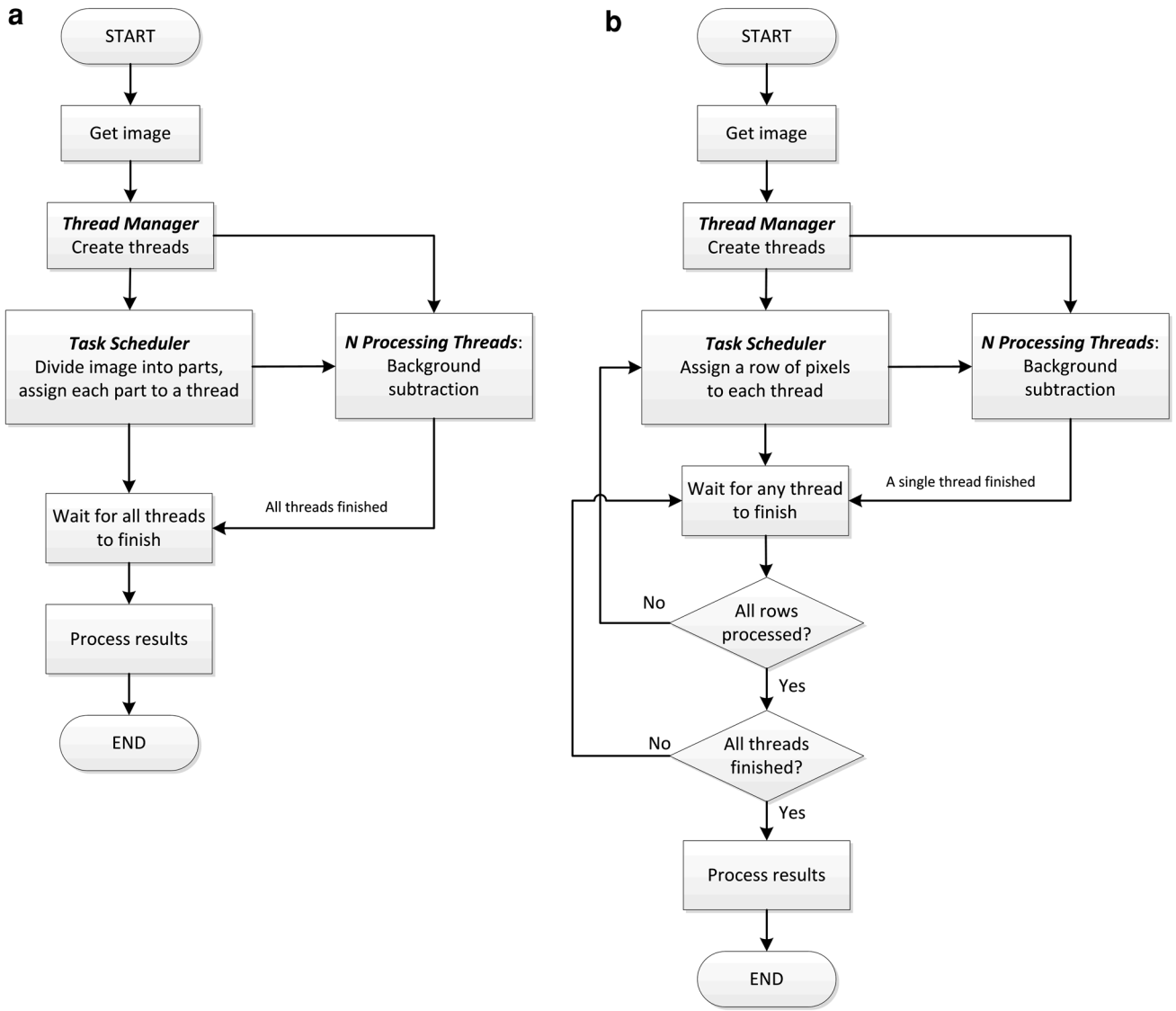

Fig. 1 Simplified block diagrams of the parallel background subtraction in image frames using **a** the static and **b** the dynamic task scheduling methods

outer loop) and then each row by pixel (the inner loop), it was decided to parallelize the outer loop iterations (*pragma parallel for*), so that a single row of pixels is assigned to a thread at a given time. It was previously verified that assigning larger data chunks to threads (more than one pixel row) does not improve performance of the algorithm [7]. Moreover, the OpenMP library also provides a guided task scheduling method, which in theory should balance the advantages and disadvantages of the static and dynamic approaches [3]. However, the guided method in the current

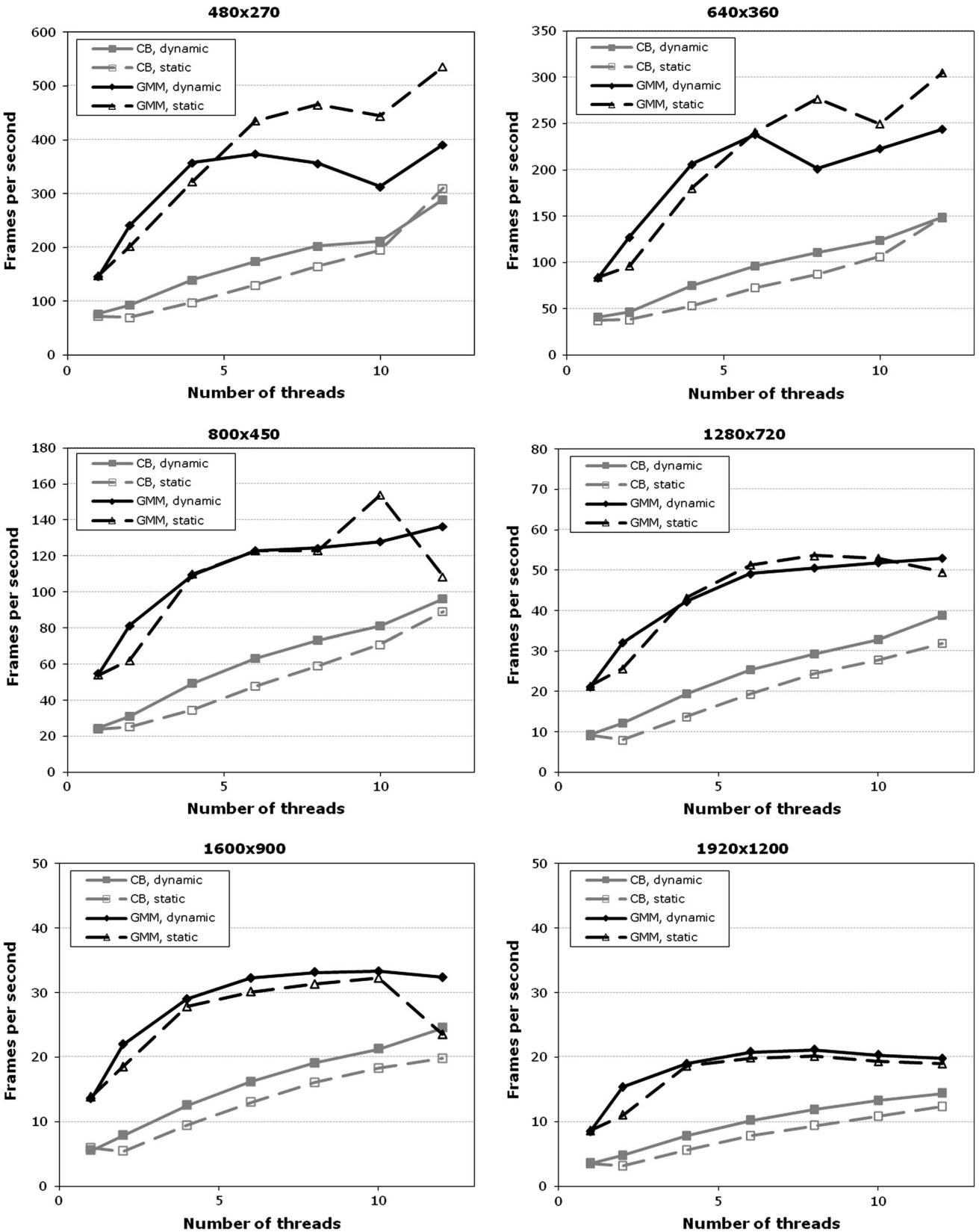

Fig. 2 Results of performance testing of the parallel GMM and Codebook algorithms, using the static and the dynamic task scheduling in OpenMP, expressed as frames processed per second versus the number of processing threads. The *Y* scale is different for each subplot in order to provide better visualization of the results

implementation of the GOMP library used for tests does not perform better than the static one [7].

The morphological processing is parallelized using the same approach. The static task scheduling method is sufficient in this case, because the processing times for different pixels are comparable. Because the output value depends on the input values of pixels in the neighborhood, each dilation/erosion operation has to be run separately, in sequence.

5 Experiments

5.1 Performance testing

The performance of the background subtraction algorithms was tested on a live 'supercomputer' system using the KASKADA framework. A single computing node contained two six-core CPUs in Xeon EM64T architecture, running at 2.27 GHz, with 16 GB RAM, controlled by the Debian Linux operating system. The whole system consists of 192 nodes connected with Infiniband network and its computing power is rated at 20.9 TFLOPS [31]. The algorithms were written in C++ language and compiled with the GNU compiler in version 4.3.2 with the parameters tuned to the supercomputer architecture. The GOMP library (an implementation of the OpenMP API) was used for the parallel task scheduling.

In the tests, a camera recording of HD resolution (1,920 × 1,200 pixels, 25 fps) and several downscaled versions of the same recordings were processed. Since it was not possible to find a suitable HD recording in the available benchmark sets, a typical recording from the city traffic-monitoring camera, with an uniform movement of vehicles in the camera frame was chosen. The initial 100 frames of the recording were used for the background model training and the actual background subtraction was timed in the remaining 1,400 frames. For each image resolution, seven values of concurrently running processing threads, from 1 to 12, were tested. The hyperthreading capabilities of the CPUs were not used, only the physical cores were employed. Two task scheduling methods: the static and the dynamic one, were evaluated using the OpenMP system. Each test run (for a given resolution, task scheduling method and number of threads) was repeated 12 times and the measured running times were expressed in a number of frames processed per second (fps). Only the background subtraction procedure was timed, other operations related to video decoding, memory buffer management, etc., were not included in measurements, because they are common for each tested algorithm in all configurations. In each test run, the lowest and the highest recorded fps values were discarded and the remaining 10 test results were averaged.

The results of the performance testing for the original GMM and the Codebook algorithms are presented in Fig. 2. The modified GMM algorithm performed similarly to the original one so it was omitted for the result clarity. Surprisingly, the observed performance of the GMM algorithm was significantly better compared with the Codebook algorithm. In many cases, the GMM algorithm was 3–4 times faster. A higher complexity of a single pixel processing, varying number of the codewords per pixel and the need for constant memory allocations and deallocations are the most probable reasons for the observed worse performance of the Codebook algorithm. The GMM method, with a fixed number of Gaussians per pixel, was faster despite the need for performing an additional shadow removal. For lower image resolutions (up to 800 × 450), a single thread is sufficient to achieve the processing speed exceeding twice the source fps value if the GMM algorithm is used. For larger resolutions, 4–6 threads are needed for similar processing fps. However, in case of the HD recording, the GMM algorithm was not able to achieve satisfactory fps value. The Codebook algorithm needed 8–12 threads in order to achieve the same performance level as the GMM with 2 or 4 threads.

In terms of the parallel processing, increasing the number of threads for the Codebook algorithm resulted in reduction of the processing time and the relation was, in most of the cases, almost linear, although the fps values obtained for higher resolutions are not satisfactory. With the GMM algorithm, increasing the number of threads gives a performance boost up to eight threads and using more than six threads is not significantly beneficial. This is especially evident for the lower image resolutions, where too many threads decrease the algorithm performance. However, this effect is observed for very large fps values

Table 2 Characterization of the dataset utilized for the algorithm quality assessment

Dataset	Video name	Resolution	Fps	Scene characteristics
PETS 2001	ds1_ts_c2	768 × 576	25	Outdoor, parking lot, low illumination variations
PETS 2001	ds3_ts_c1	768 × 576	25	Outdoor, parking lot, high illumination variations
PETS 2006	S1-T1-C3	720 × 576	25	Indoor, train station, constant illumination
PETS 2006	S3-T7-A3	720 × 576	25	Indoor, train station, constant illumination

Table 3 Object detection algorithms settings utilized during the carried out experiments

Method	Codebook	GMM	Modified GMM
Parameters	$\alpha = 0.7$	$\rho = 0.0005$	$\rho = 0.0005$
	$\beta = 1.2$	$\alpha = 0.0005$	$\alpha = 0.0005$
	$\varepsilon_1 = 20$	$k = 5$	$k = 3$
	$\varepsilon_2 = 20$	$T = 0.5$	$T = 0.5$
	$T_h = 20$	$\Delta = 2.5$	$\Delta = 2.5$
	$T_{add} = 100$		$\gamma = 0.075$
	$T_{del} = 100$		$D_{th} = 5$

(more than 150), so it has no practical implications. The most important observation is that for the HD recording, using more than four threads do not improve the processing time in a significant way. One possible explanation for the abovementioned effects is that the payload related to running and managing a larger number of threads is not balanced by the decrease in the actual processing time of a single thread. It should be noted that when the input image is divided into more than four parts, decrease in the chunk size for a single thread is not as significant as when two or four threads are used instead of one, especially for lower image resolutions. Also, when the number of the processing threads exceeds the number of the physical cores of a single CPU (which is equal to six), some hardware issues, especially related to the memory access, may influence the results. However, these details are difficult to test and the authors of this article did not have access to the low level of the system architecture, they were only able to implement the tested algorithms in a way that the internal organization of data structures or the order of pixel processing do not influence the performance of the algorithm.

Comparing the dynamic and the static task scheduling methods, it may be observed that the dynamic approach improves the performance of the Codebook algorithm, being 1.3–1.6 times faster than the static one when two to six threads are used. The difference decreases with the rising number of threads. For the low resolutions and a high number of threads, the overhead of the dynamic method is too high. For the GMM method, both approaches yield comparable results in the range of up to 200 fps values, the dynamic method is marginally better for the lower number of threads. Therefore, the static approach is sufficient in case of the GMM method, while for the Codebook algorithm the dynamic method improves its performance.

5.2 Accuracy of background subtraction

In order to evaluate the detection accuracy of both implemented methods, the experiments using a set of benchmark videos were carried out. A precise and objective evaluation of the object detection results is a difficult task. Most

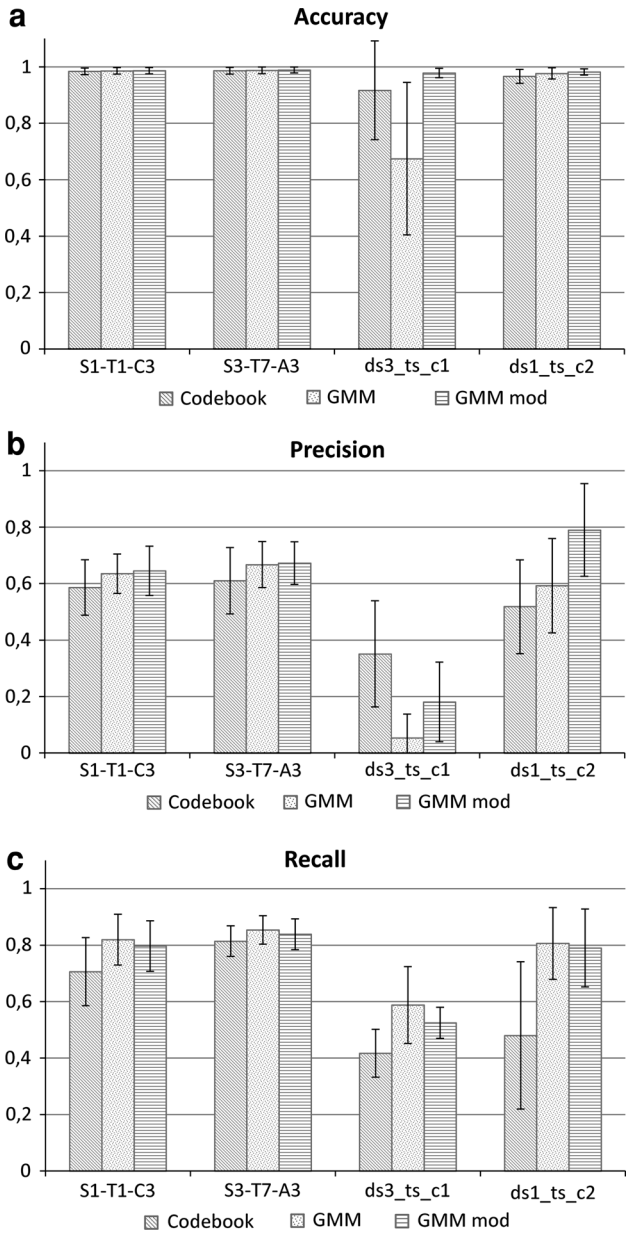

Fig. 3 Results of object detection quality testing of the Codebook, GMM and modified GMM algorithms. Precision, recall and accuracy are expressed for each of the four recordings as a mean value with a standard deviation *bar*

authors verify the algorithms on the basis of a single selected video frame. However, techniques based on the virtual scenes can also be found in the literature [2]. In this work, the following approach was utilized. Two recordings from the PETS2001 [21] and two recordings from the PETS2006 [22] datasets were chosen. Two of the selected video samples from the PETS2006 dataset represent a typical indoor scene (S1-T1-C3, S3-T7-A3) with medium intensity of the object movement and a stable lighting. Both recordings from the PETS2001 dataset depict an outdoor scene with a parking lot (ds1_ts_c2, ds3_ts_c1).

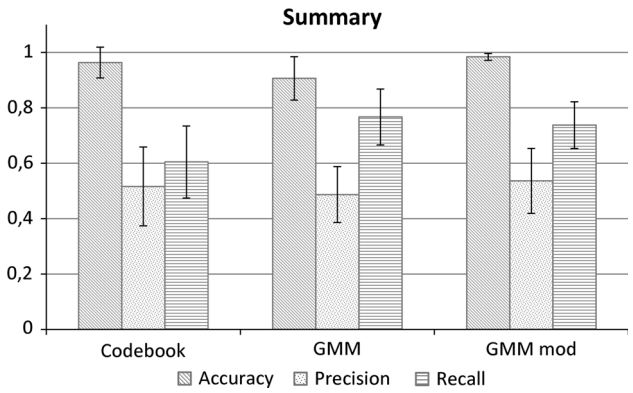

Fig. 4 The summary of object detection quality testing of Codebook, GMM and modified GMM algorithms. The averaged results for all the analyzed recordings are presented

The second of these videos represents difficult conditions for object detection: illumination variations caused by the clouds moving on the sky on a sunny day. Such a dataset allows for evaluating the algorithms in different conditions. The parameters of the test videos are presented in Table 2.

For the purpose of assessment of the object detection accuracy, the ground truth data was prepared. In order to obtain it, a set of images in 60 frames intervals was extracted from each of the analyzed recordings. These images were then manually processed and, as a result, the masks denoting the foreground objects were created. This way, 223 images representing the ground truth data were acquired.

For the assessment of the object detection results, the following three measures based on errors of type I and II were calculated:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{11}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{12}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}, \tag{13}$$

where TP pixels correctly assigned to the foreground (true-positive result), TN pixels correctly assigned to the background (true-negative), FP pixels incorrectly assigned to the foreground (false-positive), FN pixels incorrectly assigned to the background (false-negative).

The precision factor denotes the rate of the correct detection results in relation to the whole foreground area. The recall parameter indicates the degree of the relevant detections. The accuracy is a more comprehensive measure, which shows the overall similarity of the result to the ground truth data. All of these parameters should be maximized for the optimal algorithm. These factors were calculated for each of the prepared ground truth data.

In order to acquire the object detection results from the implemented algorithms, all four test recordings were analyzed by three algorithms (the original GMM, the modified GMM and the Codebook). Identical algorithm parameters, listed in Table 3, were used for each of the processed videos. During the preliminary experiments, various parameter values were tested. The chosen settings allow for achieving a compromise between the results acquired for all four analyzed recordings.

The aggregated results for each of the recordings are illustrated in Fig. 3. Analyzing the precision plot, it can be observed that for all the four videos, the modified version of the GMM method gives the best results in terms of the mean precision and the result stability. Additionally, the high precision stability related to low deviation of the measure for the utilized benchmark data is achieved. A high precision value means that more pixels were correctly assigned to the foreground and at the same time, fewer pixels were incorrectly excluded from the background. For the adjusted GMM method, this outcome is related to better adaptation capabilities. Hence, the static image regions observed under the changing illumination were still recognized as the background elements. The most significant differences between the regular and the modified GMM method (in the favor of the second) are noticeable for the recording with high lighting variations.

The recall plot depicts the rate of how many of the relevant results were correctly denoted. In other words, it is related to the sensitivity of the algorithm. Hence, the higher the recall is, the more objects are reliably detected. The results show that the regular GMM method is the most sensitive of all the evaluated object detection algorithms. This outcome agrees with the predictions regarding the modified GMM approach since the adjustments were designed to lower the algorithm sensitivity. The Codebook method turns out to perform worst, similarly as for the precision factor. However, in the case of the outdoor recording with illumination changes, it outperforms the regular GMM algorithm significantly.

The accuracy factor is an overall score which describes the rate of correct classifications to both categories, which in this case are the background and the foreground. In terms of the accuracy factor, all the assessed algorithms produce comparable results. Only for the recording where the dynamic illumination changes are present, relevant differences can be seen. The mean accuracy value for the Codebook and for the modified GMM method are similar. However, the Codebook algorithm produces less stable results.

The optimal object detection algorithm should be characterized by maximized values of the precision, recall and accuracy factors. Hence, a summary of the results with the

scores from each processed video aggregated is presented in Fig. 4. Analyzing the overall score, it can be stated that the best results for the prepared dataset in terms of the object detection quality can be achieved with the modified GMM method, followed by the regular GMM algorithm and the Codebook as the least accurate one.

6 Conclusions

The task of the authors was to implement the background subtraction algorithm in a parallel manner, within the framework for a complex multimedia stream processing running on a supercomputer. In order to realize this task, a sufficiently accurate algorithm, a method of parallel implementation and a strategy of assigning the image pixels to the threads had to be chosen. The authors have not found such a research in the published works, therefore the results, which may be useful for other researchers implementing the background subtraction procedure on a parallel system, were presented in this article. The authors also proposed an improvement to the original GMM algorithm making it more robust in difficult lighting conditions and also a modification of the training phase in the Codebook algorithm.

Two background subtraction algorithms were tested—the GMM (in two versions: the original one and the proposed modification) and the Codebook—and the accuracy and performance of their parallel implementation on a supercomputer platform for the real-time video processing were evaluated. The GMM algorithm proved to be significantly more efficient than the Codebook. For the high-resolution images, the rate of processed frames per seconds could not achieve the source fps rate, despite increasing the number of processing threads running concurrently on the separate physical processing units. For the lower image resolutions, two to six threads provide a sufficient performance, and increasing the number of threads above this level does not reduce the processing time. The Codebook algorithm is too slow for a practical application in the real-time video analysis system. This method has some potential, but the algorithm requires reworking in order to reduce the complexity, especially regarding the memory management procedures.

In terms of the object detection accuracy, all tested algorithms provide a satisfactory level of the correct results. When all three measures—the overall accuracy, precision and recall—are taken into account, the GMM algorithm provides the best results for both the indoor and outdoor videos, and it outperforms the original GMM method in case of difficult lighting conditions.

On a basis of the performed efficiency and accuracy tests, the GMM algorithm with the proposed modification was selected for the parallel implementation on a supercomputer. Because the conducted research employs mainly cameras of PAL (768 × 576) and higher resolutions, six physical CPU cores for six processing threads should be sufficient for obtaining the processing fps rate at a level suitable for the real-time processing, without the need for decreasing resolution or dropping frames. An automatic thread management with OpenMP API is used, because it provides good performance and reduces the amount of work related to adapting the algorithm for parallel implementation. It was also decided to use the dynamic task scheduling, as the overhead related to the resource management is not significant and the dynamic approach may be useful in case of camera scenes with movement limited to some horizontal image sections.

In the future, the authors plan to test the performance of the background subtraction algorithms together with other video processing algorithms, such as object tracking, classification and event detection, running on other supercomputer nodes and communicating with a message passing interface system. The future work will also focus on processing high-resolution images using multiple supercomputer nodes in order to avoid the observed effect of saturated performance of the algorithm. Additionally, the authors plan to test the performance of the background subtraction algorithms implemented on GPU devices in order to compare the results with these presented in this article and to asses the capabilities of the modern GPU devices for the real-time processing of high-resolution video streams from cameras.

Acknowledgments Research funded within the project No. POIG.02.03.03-00-008/08, entitled "MAYDAY EURO 2012—the supercomputer platform of context-depended analysis of multimedia data streams for identifying specified objects or safety threads". The project is subsidized by the European regional development fund and by the Polish State budget.

References

1. Benezeth, Y., Jodoin, P.-M., Emile, B., Laurent, H., Rosenberg, C.: Comparative study of background subtraction algorithms. J. Electron. Imaging 19(3) (2010). doi:10.1117/1.3456695
2. Brutzer, S., Hoeferlin, B., Heidemann, G.: Evaluation of Background subtraction techniques for video surveillance. In: Proceedings of 24th IEEE Conference on Computer Vision and Pattern Recognition, 1937–1944 (2011)
3. Chapman, B., Jost, G., van der Paas, R.: Using OpenMP. Portable shared memory parallel programming. MIT Press, Cambridge (2007)

4. Chen, G., Yu, Z., Wen, Q., Yu, Y.: Improved Gaussian mixture model for moving object learning. Artif Intell Comput Intell LNCS **7002/2011**, 179–186 (2011). doi:10.1007/978-3-642-23881-9_23
5. Czyżewski, A., Szwoch, G., Dalka, P., et al.: Multi-stage video analysis framework. In: Lin, W. (ed.) Video Surveillance, pp. 147–172. Intech (2011)
6. Dougherty, E., Lotufo, R.: Hands-on morphological processing. SPIE Press, Bellingham (2003)
7. Dziech, A., Czyżewski, A. (eds.): Performance evaluation of the parallel codebook algorithm for background subtraction in video stream. Multimedia Communications, Services and Security, Communications in Computer and Information Science, Springer **149**, 149–157 (2011)
8. Fauske, E., Eliassen, L.M., Bakken, R.H.: A comparison of learning based background subtraction techniques implemented in CUDA. In: Proceedings of Norwegian Artificial Intelligent Symposium (NAIS), pp. 181–192, Trondheim (2009)
9. Friedman, N., Russell, S.: Image segmentation in video sequences: a probabilistic approach. In: Proceedings of 13th Conference on Uncertainty in Artificial Intelligence (1997)
10. Horprasert, T., Harwood, D., Davis, L.S.: A statistical approach for real-time robust background subtraction and shadow detection. In: Proceedings of IEEE Frame Rate Workshop, Kerkyra, pp. 1–19 (1999)
11. Ilyas, A., Scuturici, M., Miguet, S.: Real time foreground-background segmentation using a modified Codebook Model. In: Proceedings of Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance AVSS, pp. 454–459, Genova (2009)
12. KaewTraKulPong, P., Bowden, R.: An improved adaptive background mixture model for real-time tracking with shadow detection. In: Proceedings of 2nd European Workshop on Advanced Video Based Surveillance Systems (2001)
13. Karasulu, B.: Review and evaluation of well-known methods for moving object detection and tracking in videos. J. Aeronaut. Space Technol. **4**, 11−22 (2010)
14. Kempf, B.: The Boost Threads library. C/C++ Users J. **20**(5), 6–13 (2002)
15. Kim, K., Davis, L.S.: Object detection and tracking for intelligent video surveillance. Multimed Anal Process Commun **346**, 265–288 (2011)
16. Kim, K., Chalidabhongse, T.H., Harwood, D., Davis, L.: Real-time foreground-background segmentation using codebook model. Real-time Imaging **11**, 167–256 (2005)
17. Kim, K., Harwood, D., Davis, L.: Background updating for visual surveillance. In: Bebis, G., Boyle, R., Koracin, D., Parvin, B. (eds.) Advances in Visual Computing, LNCS, vol. 3804, pp. 337–346. Springer, Heidelberg (2005)
18. Krawczyk, H., Proficz, J.: KASKADA—Multimedia processing platform architecture. In: Proceedings of International Conference on Signal Processing and Multimedia Applications (SIGMAP), Athens, pp. 26–31 (2010)
19. Li, Y., Chen, F., Xu, W., Du, Y.: Gaussian-Based Codebook Model for video background subtraction. Advances in Natural Computation, LNCS, vol. 4222, pp. 762–776, Springer (2006)
20. Parks, D.H., Fels, S.S.: Evaluation of background subtraction algorithms with post-processing. In: Proceedings of IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance AVSS, pp. 192–199, Sanfa Fe (2008)
21. PETS 2001 dataset.: 2nd IEEE international workshop on performance evaluation of tracking and surveillance (PETS 2001), Kauai, Hawaii (2001). http://www.cvg.cs.rdg.ac.uk/PETS2001/pets2001-dataset.html. Accessed 4 Dec 2012
22. PETS 2006 dataset.: 9th IEEE international workshop on performance evaluation of tracking and surveillance (PETS 2006), New York (2006). http://www.cvg.rdg.ac.uk/PETS2006/data.html. Accessed 4 Dec 2012
23. Pham, V., Vo, P., Hung, V.T., Bac, L.H.: GPU Implementation of extended Gaussian Mixture Model for Background Subtraction. IEEE RIVF International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), pp. 1–4, Hanoi (2010)
24. Reinders, J.: Intel threading building blocks. Outfitting C++ for multi-core processor parallelism. O'Reilly Media, Sebastopol (2007). ISBN 10: 0-596-51480-8. http://shop.oreilly.com/product/9780596514808.do
25. Setiawan, N.A., Seok-Ju, H., Jang-Woon, K., Chil-Woo, L.: Gaussian mixture model in improved HLS color space for human Silhouette extraction. Advances in artificial reality and tele-existence, Lecture notes in computer science, vol. 4282, Springer, pp. 732–741 (2006)
26. Shimada, A., Taniguchi, R.: Hybrid background model using Spatial-Temporal LBP. Advanced Video and Signal Based Surveillance (2009)
27. Sicre, R., Nicolas, H.: Improved Gaussian mixture model for the task of object tracking. Lecture Notes in Computer Science, vol. 6855/2011, pp. 389–396 (2011). doi:10.1007/978-3-642-23678-5_46
28. Sigari, M.H., Fathy, M.: Real-time background modeling/subtraction using Two-Layer Codebook Model. In: Proceedings of International Multi-Conference of Engineers and Computer Scientists IMECS, pp. 1–5. Hong Kong (2008)
29. Stauffer, C., Grimson, W.E.: Adaptive background mixture models for real-time tracking. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 246–252. USA (1999)
30. Suo, P., Wang, Y.: An improved adaptive background modelling algorithm based on Gaussian mixture model, ICSP (2008)
32. TASK: Galera plus. http://www.task.gda.pl/kdm/kdm/gplus (in Polish)
31. Wang, J., Dong, L.: Moving objects detection method based on a fast convergence Gaussian mixture model. In: 3rd International Conference on Computer Research and Development, pp. 269–273. China (2011)
33. Zivkovic, Z., Van der Heijden, F.: Efficient adaptive density estimation per image pixel for the task of background subtraction. Pattern Recogn. Lett. **27**(7), 773–780 (2006)

Author Biographies

Grzegorz Szwoch was born in 1972 in Gdansk. In 1996, he graduated as a student from the Sound Engineering Department. His thesis was related to physical modeling of musical instruments. Since that time he has been a member of the research staff at the Multimedia Systems Department. In 2004, he received his Ph.D. degree. His Ph.D. thesis was devoted to acoustic modeling of hearing aid waveguides. Currently, he participates in research projects concerning intelligent video monitoring algorithms. His work in this area focuses on object detection and tracking, and on automatic detection of important security threats in video.

Damian Ellwart was born in Gdynia in 1984. He completed his studies at the Electronics, Telecommunication and Informatics Department of Technical University of Gdansk in 2008 acquiring M.Sc. degree. The subject of his thesis was recognition of moving

vehicles on the basis of image analysis. Currently he is a Ph.D. student of Multimedia Systems Department. He takes part in research projects related to automatic video monitoring, focusing on object classification, route reconstruction and privacy enhancement.

Prof. Andrzej Czyzewski is a native of Gdansk, Poland. He received his M.Sc. degree in Sound Engineering from the Gdansk University of Technology in 1982, his Ph.D. degree in 1987 and his D.Sc. degree in 1992 from the Cracov Academy of Mining and Metallurgy. Prof. Czyzewski joined the staff of the Sound Engineering Department of the Gdansk University of Technology in 1984. In December 1999, Mr. President of Poland granted him the title of Professor. In 2002, the Senate of his University approved him to the position of Full Professor. He led a number (more than 10) of research projects sponsored by the Polish State Committee for Scientific Research and two European projects. He is the author of more than 300 research papers published in international journals and presented in congresses and conferences around the World. He is also author of eight Polish patents in the domain of computer science and four international patent applications in the domain of telemedicine.